Different Path Planning Approaches for Autonomous Mobile Robots

Prajjwal Dutta
School of Electronics and Communication Engineering
Vellore Institute of technology
Vellore, India
prajjwaldutta2001@gmail.com

Dr. Sujatha R
School of Electronics and Communication Engineering
Vellore Institute of Technology
Vellore, India
sujatha.r@vit.ac.in

***Abstract*—Robotics is the future of technology. Statistically, most industrial companies are in the process of installing robotics systems and technologies in different kinds of sectors. Moreover, nowadays robots without human intervention have come into the highlight and have been in high demand from not only autonomous vehicle industries but also from other different kinds of corporate industries. For that reason, different kinds of path-planning algorithms are needed for different kinds of requirements according to industrial needs. So, finding the most efficient algorithm among the existing path planning algorithms and tuning the parameters to their maximum efficiency is going to be the key to meeting the requirements of the robot.**

Keywords—Robotics, Autonomous Vehicle, Path Planning, Dijkstra algorithm, Heuristic search, Cost map, Gradient

I. INTRODUCTION

Autonomous vehicles are the most lucrative and promising upgradation of stereotypical vehicle technology. The application of path-planning algorithms in intelligent autonomous vehicles has significantly improved the experience of the drivers. Moreover, the latest development and instalments of different kinds of cruise control modes on vehicles help and contribute to more advanced safe driving helps in populated areas [1]. The usage of autonomous robots and vehicles has grown rapidly in the past few years. For example, different vehicle manufacturing companies (like Tesla, Tata etc.) have already introduced and launched their autonomous vehicle products in the market. Tesla has even launched their autopilot technology in their autonomous vehicles. Automated robots and UAVs are also upcoming emerging technology that is being industrialized very rapidly. For example, many restaurants in Japan have started using autonomous robots for serving food to customers. Autonomous moving robots are used for delivery services. Also, amazon is now commercializing the usage of drones for door-to-door package delivery. Path planning is also used for autonomous underwater vehicles to complete their required tasks [2]. The main basis of all these different applications of an autonomous mobile robot is path planning and obstacle avoidance. Different path-planning approaches are used for different kinds of requirements of the robots. In this paper, we have tried to use different kinds of path-planning algorithms with different goal search patterns. Every algorithm has its pros and cons and is suitable for different kinds of applications.

II. LITERATURE REVIEW

According to N.Sariff and N. Buniyamin, "Simulation and experimental results from previous research show that algorithms play an important role to produce an optimal path (short, smooth and robust) for autonomous robot navigation and simultaneously it proves that appropriate algorithms can run fast enough to be used practically without a time-consuming problem." [3]. So, the existing algorithms are being modified as per the applications of mobile robots to get the desired output.

"The planning module of an autonomous or self-driving vehicle should ensure safety and comfort for the passengers. It should also put the vehicle in the right behaviour concerning the kinematic and motion model constraints surrounding the vehicle." (Christos Katrakazas, Mohammed Quddus) [4]. According to their theory, the three most important aspect of the path planning approach is finding a path, searching for the safest way without any obstacles and determining the most efficient path.

Different kinds of algorithms are innovated and tested for different kinds of scenarios. Dijkstra, Best First Search algorithm (BFS), A* algorithm, and SLAM algorithm for mobile robot path planning are some of the basic algorithms for path planning. For example, Buniyamin N. and Wan Ngah W.A.J. in their research have used a bug algorithm family which is a local path planning algorithm. According to them this algorithm "uses sensors to detect the nearest obstacle as a mobile robot moves towards a target with limited information about the environment. The algorithm uses obstacle border as guidance toward the target as the robot circumnavigates the obstacle till it finds a certain condition to fulfil the algorithm criteria to leave the obstacle toward the target point." [5]

Sometimes, the default path planning algorithms are modified to create a dynamic path planning algorithm. Adem Tuncer and Mehmet Yildirim have invented a parallel global search technique called a genetic algorithm. It emulates natural genetic operators. [6]

Besides these, the localization (initial position of the robot) and the goal position are very important to be known for any kind of path-planning approach. The robot must be aware of its initial position and the environment around them. Mark Pfeiffer and Michael Schaeuble have represented a 2-D model which learns a complex mapping from the laser sensors in the robot and it is a target or goal-oriented end-to-end navigation model. [7]

Genetic algorithm is used vastly for obstacle avoidance and gives the most optimized path for goal-to-goal robot navigation. This algorithm works in both known and unknown environments. So, it's an upgraded version of the SLAM algorithm for an intelligent autonomous robot. [8]

One of the most widely used path-planning algorithms is the A* search algorithm. Given Eric A. Hansen and Shlomo Zilberstein, the algorithm finds the goal state through a sequence of actions, starting from the initial state of the robot and travelling till it reaches the goal position. [10]. This algorithm has been used for mobile robot path planning from its initiation times. But gradually much research has helped to improvise the algorithm by introducing different modifications as well as tuning the parameters, to get a more accurate and faster path towards the goal location.[11].

The Rapidly-exploring Random Tree (RRT) algorithm is another path-planning approach for faster response. This algorithm is a probabilistic search approach. RRT algorithm is used for larger maps. According to Nik A, "The particle RRT algorithm explicitly considers uncertainty in its domain" [12]. Also, Haojian Zhang and Yunkuan Wang have improved this RRT algorithm for high-dimensional robotic manipulators because, in complex scenarios, the existing RRT algorithm often fails at local minima.[13]

Another real-time robot path planning algorithm is discussed by Prahlad Vadakkepat, Kay Chen Tan and Wang Ming-Liang, called the Artificial Potential Field method. It is a combination of genetic algorithms to find the optimal potential field.[14] The potential field algorithm is capable of finding the slope of each grid in the map. It tries to find the slope with the lowest value on the map to find the goal location.

For many specific uses, many algorithms are clubbed together to get the desired outcome in various applications like UAVs, Rovers, Obstacle Avoiding robots [16][18], Underwater Robots etc. For example, Hongqiang Sand and his fellow researchers have discovered a hybrid path planning system based on an improved version of the A* search algorithm and used it together with an artificial potential field algorithm for unmanned surface vehicle formation.[15]

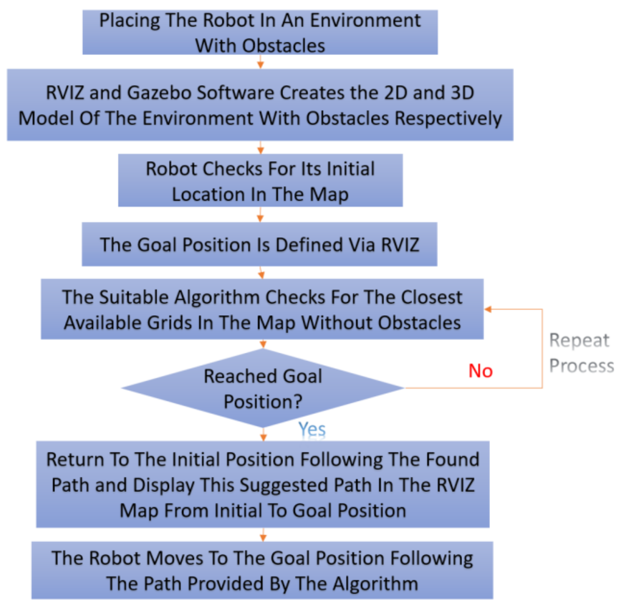
So, this is how different path-planning strategies are used in various cases for the navigation of mobile robots. Some algorithms are used for the 2-D motion planning algorithms. Some of them are evolved to be used for three-dimensional or more complex motion planning. As B.K Patel and the co-authors have mentioned that there are several classical approaches like cell decomposition (CD), roadmap approach (RA), artificial potential field (APF); reactive approaches such as genetic algorithm (GA), fuzzy logic (FL), neural network (NN), firefly algorithm (FA), particle swarm optimization (PSO), ant colony optimization (ACO) etc. [17].

III. Methodology

In this paper, an obstacle-avoiding Turtle Bot is simulated through ROS, RViz and Gazebo environment. The simulated robot is capable of avoiding any kind of obstacles in front of it and moving autonomously until it is manually stopped. At this point, the robot can move autonomously but it doesn't have a finish point. In a closed wall environment, filled with obstacles, the robot will move infinitely in the same path. Here comes the requirement of path planning approaches. In this paper, we have discussed three kinds of path-planning approaches. The algorithms generally work similarly, each with some different changes and extra classes for a different types of approaches for different applications.

For path planning, the robot needs to know its current location which is called localization. After that, it needs to know its final destination. This is called a goal position. Finally, the algorithm tries to travel along the environment (known or unknown) to reach the goal position and returns a suitable path according to the requirement and application of the robot.

A. Flow Chart


Placing The Robot In An Environment With Obstacles
RVIZ and Gazebo Software Creates the 2D and 3D Model Of The Environment With Obstacles Respectively
Robot Checks For Its Initial Location In The Map
The Goal Position Is Defined Via RVIZ
The Suitable Algorithm Checks For The Closest Available Grids In The Map Without Obstacles
Repeat Process
Reached Goal Position?
No
Yes
Return To The Initial Position Following The Found Path and Display This Suggested Path In The RVIZ Map From Initial To Goal Position
The Robot Moves To The Goal Position Following The Path Provided By The Algorithm

B. Algorithms

I. <u>Dijkstra Algorithm</u>

Previously the mentioned flowchart gives a basic idea of how the generic path planning algorithms work. But different kinds of algorithms are used for path planning depending upon many factors.

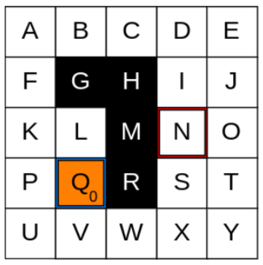
- First of all, we have used the Dijkstra algorithm. This is the most basic and initial algorithm for the maximum kinds of path planning algorithms. In this case, the algorithm is mainly focused on finding the shortest path to the goal position.
- After the map is divided into grids, the obstacles and free spaces of the grid are marked differently. The algorithm selects the grid of the robot's starting position as the initial node.
- It traverses throughout the map avoiding the obstacle grids to reach the grid of the goal position called as goal node. It follows the following equation to calculate the grid distances using the coordinate points.

$$d(x, y) = d(x) + c(x, y) < d(y) \ldots eq(1)$$

- For example, we take a map with a 5 x 5 grid with obstacle grids in it. In this map, Q_0 is the initial node and N is the goal node (Fig1).
- According to the algorithm as Q_0 is the current node as it is the initial position of the robot. The main idea of the Dijkstra algorithm is that it always keeps track of the shortest distance from the initial node to each cell of the grid. Then the node is given a value called g_cost and it is displayed in the grid's lower right corner.
- Before the search starts, the starting point Q_0 gets a g_cost 0 (as its distance to itself is 0). Later, as the algorithm progress towards the goal position, these values get updated to the actual shortest distance from the start node.
- Also, the Q_0 is put into an open list. Nodes that are inside the open list are shown in orange color.

- The algorithm then traverses to the immediate neighbor grids of the current node. It does not traverse to the neighbor grids occupied by obstacles. Then the g_cost values of these grids are updated and the previous node becomes their parent node. While moving to the neighboring grid it is given a step_cost value. This value is added to the g_cost and gets updated.

- By this technique the algorithm traverses to the neighboring nodes of these new nodes. In this example, the neighbors of the current node Q are P, L, V. then the g_cost is updated. Then the parent node is selected as from which node the g_cost is updated. The parent node of every grid is denoted in the bottom right corner of the grid. As for nodes P, L, and V, the parent node is $Q_{0.}$ The 0 of q is added with 1 and the total sum 1 is given as the g_cost value of P, L, and V. finally these neighboring nodes are added to the open list.



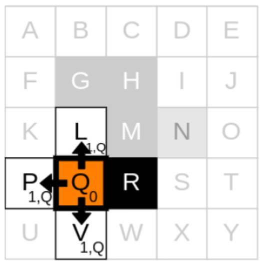


Fig1. 5x5 grid map with obstacles, Fig2. Selecting initial node Q, Finding neighboring nodes, Providing the selected nodes with g_cost values and parent node

- At the beginning of a new iteration cycle, a new current node is needed to explore. So, firstly a node is picked from the open list. It has to be one node with the smallest g_cost value. In this example, it can be any one of the nodes P, L, or V since all of them have the same g_cost value of 1. So, any of them can be picked like L and it is set to be the current node.

- Now, the previous process is repeated by updating the neighbour of L avoiding the neighbors occupied by the obstacles and the parent node (in this case Q). After this iteration, we only get K as a neighbor of L.

- Finally the g_cost value and parent node of K are updated ($K_{2,\ L}$). Adding 1 (the g_cost value of L) with 1 (the step_cost when moving from L to K) to obtain 2 and the parent node of K is L.

- Finally, K is added to open_list. To end this process finally L is added to the closed_list.

- The same process can be iterated for V (U and W).

- Finally, at the time of P all of its neighboring nodes are visited before as all of them are either in an open or closed list. So, without any process P is updated to the closed list (*Fig 3*).

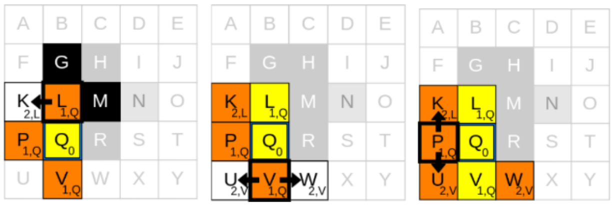
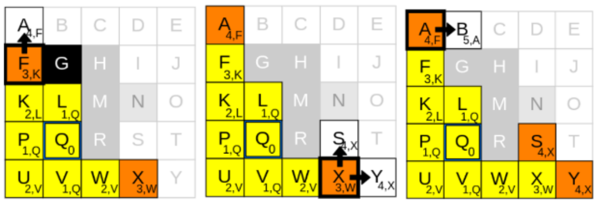


Fig3. Finding and updating the values of the neighbors of L, V, and P

- Now for our third iteration we select the current road as W. The only free neighbor is X. its g_cost value is updated to 3 (2+1=3). W is set as the parent node of X ($X_{3,\ W}$). we add X to the open list and W to the closed list. U is another non-visited node, as it has no further steps other than adding it to the closed list

- Then F is selected as a new current node and according to the figures below the algorithm proceeds.



- In this iteration let S is selected as the current node and its neighbors are checked. The available neighboring grids are N and T. for each of these g_cost values and the parent node is set. They are stored in the open list and S is stored in the closed list.

- Then Y is set to the current node and its available neighbor is T. Its g_cost value is also calculated but as this node already has a g_cost value from S neighbor, the new and previous g_cost values are compared. Since the new value is not lower it is discarded and Y is put into the closed list.

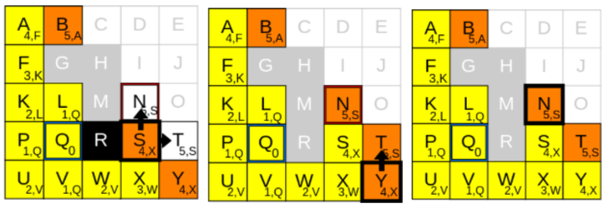


Fig4. Continuing the iteration until the algorithm reaches the goal node (N)

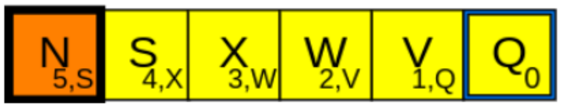
- Finally, N is selected as the current node (*Fig 4.3*), and as it is equal to the goal/target node the algorithm stops scanning for more nodes. The exploration of the map and collection of the data is complete.

- The second part of the algorithm is to utilize the collected data to build the shortest path.

- Since the target node is reached by the algorithm, the node is taken and added to a new list. Then the algorithm looks for its parent node (S in the case of N) and adds it to the list.

- In the same way the parent's parent nodes are discovered and stored in the list. This iteration continues until the initial node Q_0 is reached, which has no parent node. This process is called back-tracking.

- Once the process is complete, the newly created list will contain the shortest path as a sequence of grid cells to visit, but in the wrong order.



- To find the right shortest path the list needs to be reversed.

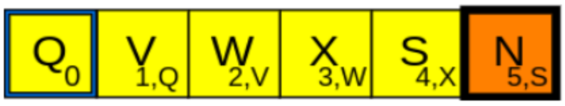
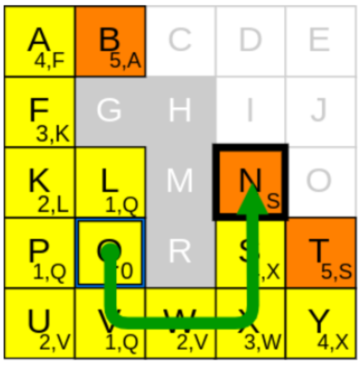


Fig5. Reversed grids after backtracking for the shortest path

- Thus the path planning process completes and the shortest path from the initial to the target position for the robot is discovered using the Dijkstra algorithm.



- An Rviz simulated environment is created (*Fig6*) and the Dijkstra algorithm is implemented.

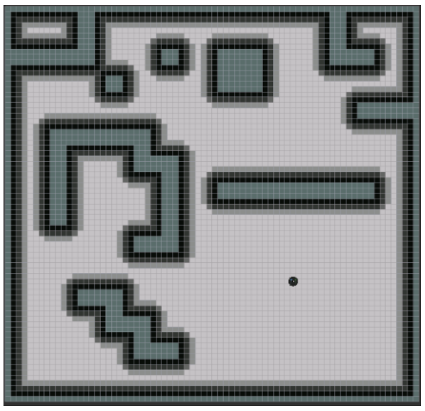


Fig6. Rviz simulated a 2-D grid map

- The algorithm traverses through all the grids and finds the suitable shortest path for the Turtle Bot after the goal position is given by the user (*Fig 7*).

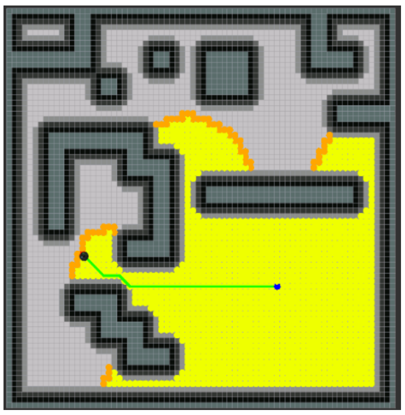


Fig7. The Turtle Bot reaches the goal position given to it following the shortest path provided by the Dijkstra algorithm

- But this algorithm has some limitations.

1. It is a uniform search or blind search algorithm. "It means that during the expansion process, it doesn't know if one grid cell is better than the other. This causes the algorithm to search in all directions in a radial pattern originating from the starting node." [9]. So, this algorithm is slower than other algorithms.
2. As it uses a large number of nodes it needs more storage in the RAM.
3. Also, for this algorithm the environment must be static with obstacles in a constant position. For a Dynamic environment, the recalculation of the path will be and the process will be even slower.

For these limitations, new advanced algorithms are tried to reduce these constraints.

II. Greedy Best First Search (BFS) Algorithm

- To solve the slower rate of the path planning algorithm the uniform search algorithm is replaced by an Informed search algorithm. In this case, the algorithm uses the information about the goal position to guide the search towards the target. It produces a more efficient form of the map [9].
- In this case, finding the direction of the goal is another big task. For that, exploration in all directions can be avoided by using polarizing methods. That's where the heuristic approach for approximating the distance to the goal position comes into play.
- In this algorithm we have used Euclidean and Manhattan distance estimators to implement the algorithm.
- The Euclidean distance (*Fig8.1*) between the initial and the goal position grid is calculated by this formula [eq (2)].

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \text{ ...eq (2)}$$

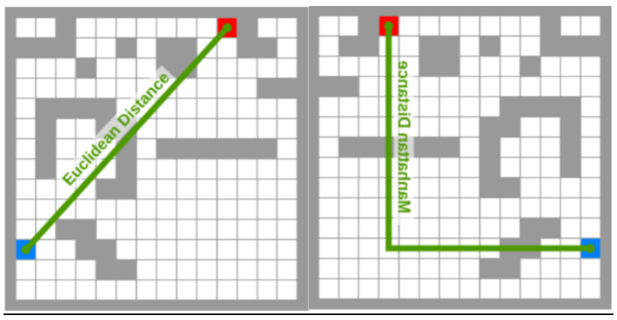


Fig8. Euclidean and Manhattan Distance

- The Manhattan distance (*Fig8.2)* is calculated by:

$$d = |x_1 - x_2| + |y_1 - y_2| \text{ ...eq (3)}$$

In both cases (x_1, y_1) is the initial starting position of the robot and (x_2, y_2) is the goal position coordinates. D is the heuristic distance between the initial and goal position [eq (3)].

- After calculating these distances using heuristic approaches the algorithm traverses through the grids using the same Dijkstra algorithm technique. But at the time of calculating the g_cost values (h_cost in the case of the BFS algorithm), the algorithm checks for the distance from the grid to the Euclidean or Manhattan distance line. Thus, polarized travelling through the grids can be achieved.

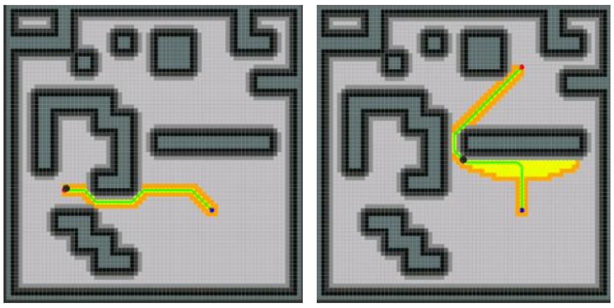


Fig9. BFS algorithm for pathfinding to the goal position by travelling through fewer grids using the heuristic distance approach

- Using this algorithm reduces the time to find the goal position and the memory requirement due to travelling through a smaller number of nodes compares to the Dijkstra algorithm.
- But there are some cases where this algorithm doesn't provide the shortest distance to the goal location. For example, in the case of concave obstacles, it results in a longer distance than the optimum shortest one. This issue arises as the algorithm tries to follow the heuristic distance lines till the position it can follow the path close to those lines (*Fig 9*).

III. A* Search Algorithm

- To solve these problems the best option was to combine both these previous algorithms. This collective algorithm is called A* algorithm.
- In industrial robotics with a static environment for mobile robots this algorithm is generally used with a few modifications according to the applications.
- In this case also the basic Dijkstra algorithm is used to find the path and the heuristic approach for the BFS algorithm is taken into account to polarise the search algorithm towards the goal location.
- But at the time of giving the cost values to the grid cells the total grid costs are called f_cost which is an arithmetic sum of the g_cost of the Dijkstra algorithm and the h_cost of the BFS algorithm.
- Then the f_cost values are used to calculate the traceback path of the algorithm from the goal to the initial node. Thus, this algorithm gives the most accurate, shortest, and optimum path to the goal position of a mobile autonomous robot (*Fig 10.2*).

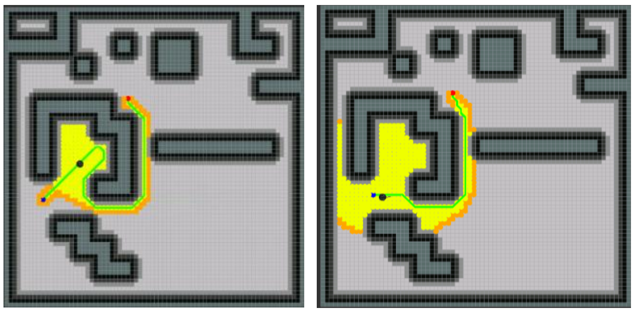


Fig10. Difference between the path planning of BFS(Fig10.1) and A (Fig10.2) algorithm*

IV. Rapidly – Exploring Random Tree (RRT)

A* algorithm is rapidly and widely used for path-planning algorithms since its introduction. However, one caveat is that it does not scale well in terms of both map size and dimensionality. As the size of the map increases, more resources (computational time, planning time) are needed. For this reason, the use of this algorithm is limited to 2-D mobile robots and robotic arms with a lower degree of freedom (DOF).

Rapidly-Exploring Random Tree (RRT) is an algorithm from a completely different family of Probabilistic Path-planning algorithms. The algorithm from this kind of family is very popular for solving complex, high-resolution, and high-dimensional path planning algorithms.

Instead of directly connecting grid cells, sampling-based algorithms generate candidate waypoints at completely random positions in the map and then examine, if these positions can be connected without hitting the obstacles. RRT is one such algorithm. It is regarded as the most efficient tool for robotic path planning in higher dimensions.

With an example map, the algorithm can easily be described.

- At first RRT algorithm pics a random point (random point 1) in the map.
- Then it wants to connect the point to the root node (initial position node). But as the chances of crossing an obstacle or just travelling too far in the wrong direction increase with a long distance from the root node, a maximum distance of a branch length is specified, by which a new node can be away from an existing one.
- If the random point is further away than the allowed branch length it can't be joined but another node is created (node 1) in the direction of the random point with the distance of maximum branch length and connected to the root node (*Fig 11*).

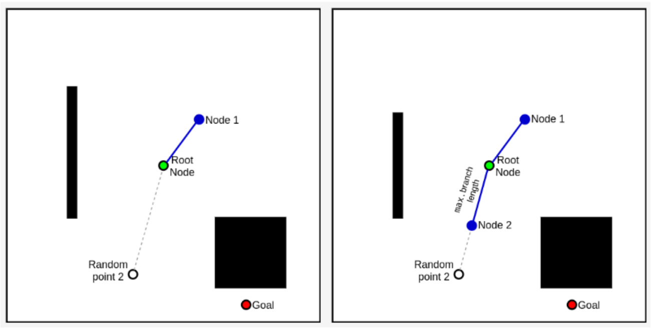


Fig 11. Randomly selecting new nodes and joining them to the nearest node

- This is how the algorithm iterates through and joins new random nodes to its nearest previously joined nodes.
- If a line that connects a new random point with its closest node, crosses an obstacle (*Fig 12.1*), the new node is discarded and the current iteration cycle is cancelled. A new iteration cycle starts which again generates another random point (*Fig 12.2*).

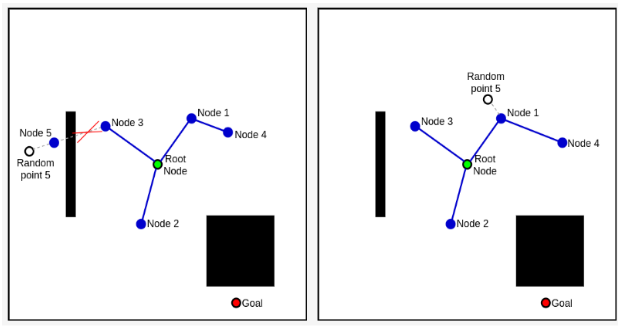


Fig12. Case of obstacles

- Thus, the expansion of the nodes continues until a connection to the goal node is found or the maximum number of iterations is exceeded. Since it is very unlikely that a random point gets generated exactly at the goal location, a certain distance from the goal (tolerance distance) is specified which is sufficient for the algorithm to declare success and stop the search.
- The final step to reconstruct the path between the start and the goal locations is a similar approach to the Dijkstra algorithm of finding parent nodes from the goal node to the root node and reversing it is applied to get the desired path from the initial to the goal position via RRT algorithm. The orange line in the fig shows the constructed path (*Fig 13*).

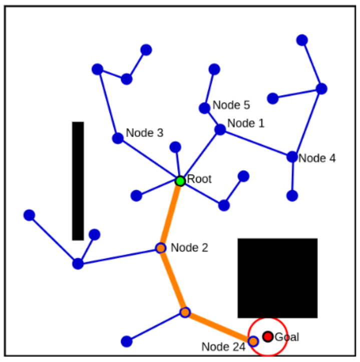
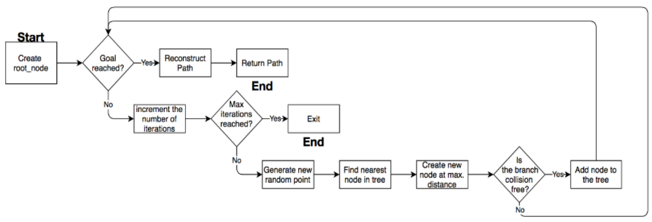
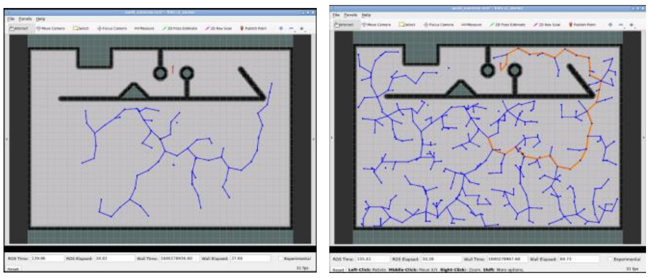


Fig13. Path Planning through RRT Algorithm

<u>Flowchart of RRT algorithm</u>


Start
Create root_node
Goal reached?
Yes
Reconstruct Path
Return Path
End
No
increment the number of iterations
Max iterations reached?
Yes
Exit
End
No
Generate new random point
Find nearest node in tree
Create new node at max. distance
Is the branch collision free?
Yes
Add node to the tree
No

<u>Model Simulation of RRT algorithm</u>



V. <u>Artificial Potential Field</u>

The most desired qualities in a path-planning algorithm are how fast and accurately the algorithm can produce a route. The higher the planning rate, the better the robot can react to changes in the environment and perform on large maps.

Similar to the RRT algorithm, the artificial potential fields or virtual potential fields are also developed precisely and used for very fast response, a direct approach for guiding a robot towards a goal location while keeping a safe distance from the obstacles.

The basic idea of this approach is to model the map into a mathematical function f (x, y). it has high values near obstacles and has a minimum value at the goal location. The gradient or slope of this function is used to build a path to the goal. In our illustration, we have visualized this method as some hilly terrain. The planned path will follow the direction that a ball, placed at the robot's location, would go. Due to the effect of the slope, the ball will roll "downhill" from higher to lower points until it reaches the lowest point. Thus, the path is found from the initial to the goal position by this algorithm.

The artificial potential field requires the modelling of two different types of forces.

1. An **Attractive Field** generated by the goal.
2. A **Repulsive Field** generated by the obstacles.

The total potential field is generated with both forces.

$$U_{total}(\mathrm{X}) = U_{att}(\mathrm{X}) + U_{rep}(\mathrm{X}) \quad \text{…eq (4)}$$

Here, X represents a grid map of size (x, y).

Attractive Potential Field:

The potential values in the attractive field decrease as it gets closer to the goal. There are some commonly used attractive potential fields.

➢ **Conical Potential Field:**

$$U_{att}(\mathrm{X}) = k\,||X_{current} - X_{goal}|| \quad \text{…eq (5)}$$

Here, k is the attractive potential constant. $||X_{current} - X_{goal}||$ denotes the distances between each grid cell and the goal [eq (4)]. The field is modelled by iterating over each of its grid cells and calculating the distance of each to the goal. Here also the standard Euclidean distance equation is used to calculate the distance between the current and the goal grid cells.

Perhaps the easiest way to visualize the conic potential function is a very simple side-view of the robot in a one-dimensional map. Here, the robot can only move along the x-axis, forward or backwards. Thus, the conical potential function in this one-dimensional map can be visualized as a cone whose lowest point is directly over the goal and opens upwards on both sides. Changing the value of k helps to generate a stronger or weaker attractive force across the full extent of the attractive field.

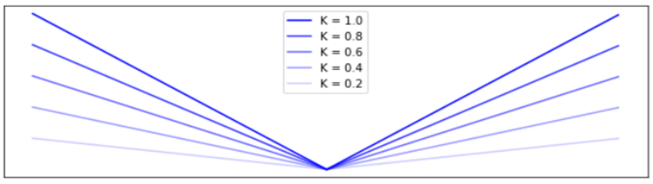


Fig14. Conical Potential Field

➢ **Quadratic Potential Function:**

$$U_{att}(\mathrm{X}) = k\,||X_{current} - X_{goal}||^2 \quad \text{…eq (5)}$$

The only difference in this function is the squared term. The functions turn into a parabolic equation [eq (5)] where the lowest point of the parabola is the goal position. Other parameters of the equation change similarly to the Conical Potential Field.

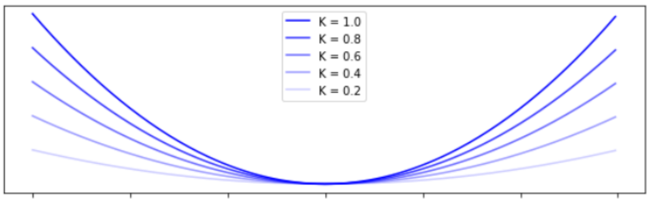


Fig15. Quadratic Potential Field

➢ **Repulsive Field Parameters:**

The basic principle is that the repulsive field pushes the robot's path away from obstacles while extending towards the goal. An ideal repulsive field must have the following qualities –

- Never allow a robot to collide with any obstacle.
- Reinforce keeping an additional safety distance.
- Have a limited range of influence.

The steps to create these repulsive fields are similar steps to designing the cost maps of ROS. Both have the values to describe the high values of obstacles and lower values towards the goal location. The repulsive field parameter is a result of two independent building blocks. The cost map of any normal map has some repulsive grids around the obstacles as a safety distance of the robot from the actual obstacles.

1. Expanding the size of map obstacles

The obstacles are expanded by at least half of the robot's width because a robot's path indicates positions where the centre of the robot's base will be moving. By expanding the obstacle at least by half of the robot's width, the path that gets into the zone of the expanded obstacle is avoided and the robot's body doesn't hit the physical obstacle. This technique helps to implement the first required quality of the repulsive field designing – "Never allow collision".

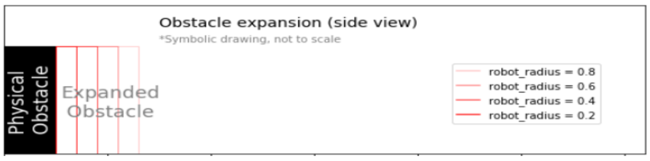


Fig16. Side view of the expansion of the obstacle

2. Creating an additional buffer zone

The buffer zone signals that it's preferable to keep an even larger distance from obstacles. This is achieved by a decaying function, whose value declines as its distance from an obstacle increases. The negative exponential function used during the implementation of the 2-D cost map in ROS is –

$$e^{k*(r-d)} \quad \text{… eq (6)}$$

Where:

- e is a mathematical constant called Euler's number (2.71828).
- k is a parameter that determines the curvature of the slope. In ROS it is called the ***cost scaling factor***.
- r is the radius of the expandable obstacle called robot radius. [eq (6)]
- d is the distance to the closest obstacle called the inflation radius. This parameter is used to set the maximum distance of influence of the buffer zone.

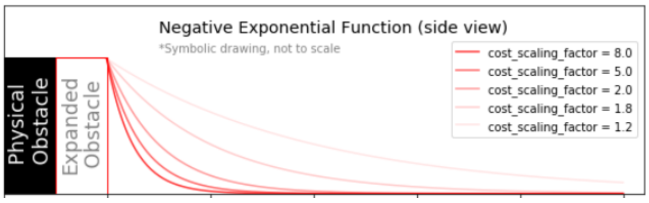


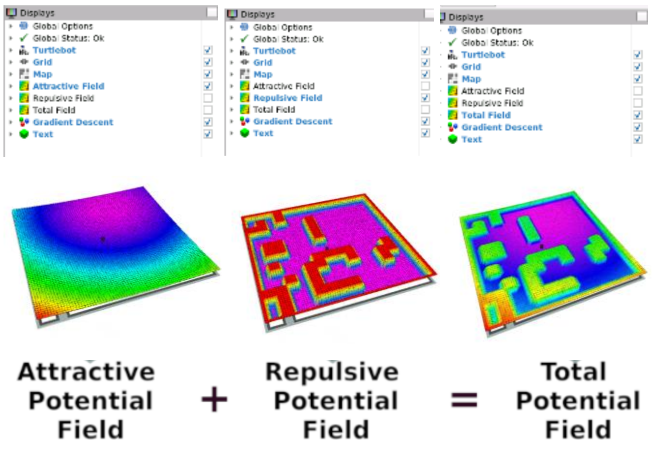
Fig17. Exponential decay of potential field away from a physical obstacle

➢ **Generating the Total Potential Field:**

The total force acting on the robot is the sum of the forces attracting the robot towards the goal position and the forces repelling the robot away from the obstacles. This can be extended to the whole map to find the Total Potential Field.

$$U_{total}(\mathrm{X}) = U_{att}(\mathrm{X}) + U_{rep}(\mathrm{X}) \quad \text{… eq (4)}$$

The resulting total potential field is represented by the 3-D surface where the lowest point in the total potential field corresponds to the desired goal location.



This is the 3-D representation of the attractive, repulsive and total potential field.

➢ **Gradient Descent**

To create the path, the calculation for gradient descent is necessary. Three main steps are repeated to calculate the steepest descent.

1. Computing the Gradient

The derivative of a function describes the slope at any given point, and its sign describes the direction of the ascent. For continuous functions, we can find its derivative using the differential formula. [eq (7)]

$$\nabla F(x,y) = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \quad \text{… eq (7)}$$

However, when working with grid maps, there are only discrete values. So, the solution is to approximate the required gradient by calculating the discrete value differences between adjacent cells. In this illustration, all eight neighbours of a grid are considered. Therefore, differences across all opposite directions are calculated to get a more accurate path.

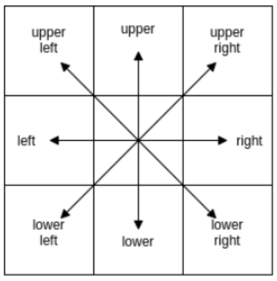


Fig18. Gradient calculation across the neighbours

2. Next Waypoint Position

After finding the discrete gradients in all directions, a grid cell located towards the direction of the steepest descent is located and picked as the next path waypoint. This process is iterated over till the final goal point is reached.

3. Creating the path

By connecting the starting position to its lowest neighbour and continuing to connect them to their respective lowest neighbours, eventually, the algorithm reaches the goal position. The algorithm adds the lowest neighbour in an empty list, each time it iterates, to create a path from the start to the goal position.

In this case, also a tolerance value is necessary as it is very unlikely to reach a grid with the lowest slope as a goal position. Also, breaking the loop is necessary to prevent the algorithm from continuing forever without getting any fruitful solution.

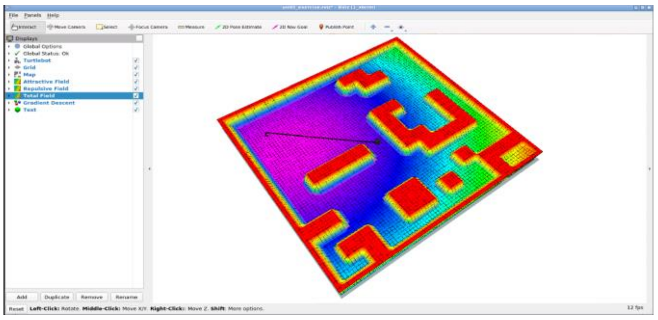


Fig19. The suggested path is drawn in black

IV. Different Uses Of The Algorithms

Algorithms	**Uses**		
	Uses	***Achievable***	***Not achievable***
Dijkstra	This algorithm can be used to find the shortest path from the initial position to the goal location. It can traverse through any kind of environment but the speed will be pretty slow with the increase in the size of the environment.	Shortest path, Unpolarized grids Exploration	Less time, Less memory
Greedy BFS	This algorithm is used to explore the grids in a particular direction rather than 360 degrees. The applications of this algorithm are best suited for an environment with long horizontal and vertical obstacles.	Polarized grid exploration, Less time to reach the goal, Less storage requires as the number of nodes is comparatively less, Uses heuristic distances to calculate the approximate distance to the goal	Shortest Path to the goal (In case of curved obstacles)
A* Search	The robot path planner can execute polarized grid exploration and reach the goal position using the most optimum path available on the map. Many autonomous robots moving in a plain area with static obstacles use this algorithm to move autonomously.	It can achieve all the cases in a static environment with the help of obstacle avoidance by the robot and provides the most efficient path.	Can't perform well in a dynamic environment with continuously moving obstacles.
Rapidly Exploring Random Tree (RRT)	The RRT algorithm is mainly a fast search algorithm. It is both based on a random and probabilistic approach.	The RRT algorithm is most useful for fast path planning. It can be used in a more complex and dynamic environment to get a faster response time.	The preciseness of the algorithm in finding the exact goal location is not as good as the other three algorithms. Most of the time it can predict the endpoint near the desired goal location but not exactly on the goal.
Artificial Potential Field	The algorithm is based on the potential field approach where the path is designed based on the gradient or slope of both attractive and repulsive potential fields. The steepest slope from starting position to the goal location is the path found by the algorithm to travel for the robot.	It can work on complex, multi-dimensional environments and uses more parameters than any of the other mentioned algorithms for making the artificial potential field model of the environment.	This algorithm is not good for global path planning this approach as it can easily get stuck in non-circular objects, objects with large shapes, L-shaped objects or dead ends or even when the goal is too close to an obstacle. It happens due to the cancellation of the attractive and repulsive forces of the field.

V. Limitations

These algorithms are strong enough to perform the path planning operations for the robot to move autonomously in the environment avoiding all obstacles. But there are some considerations for these algorithms which can be called the constraints of these algorithms.

These algorithms need to be performed under a static environment. In this kind of environment, the obstacles don't move quite often. These algorithms are powerful enough even if the environment is dynamic and the obstacles are moving frequently. But in that case, the robot will stop moving every time it faces a new obstacle that was not there at the initial phase of the path planning process and again run the complete algorithm till it reaches the goal position. This makes the algorithm slower and inefficient in a dynamic environment.

Second, the robot knows its initial coordinates. In all of these algorithms, the robot knows its initial position in the environment.

Third, the environment must be known to the robot and the robot knows the map. It is required so that the robot knows the exact positions of the obstacles on the map and plans the path accordingly.

Fourth, the robot must be able to move in any direction in 3D space including up and down. The map doesn't give an idea of the terrain of the environment. So, the robot must have the capabilities to move in any kind of terrain for the algorithm to perform perfectly.

Fifth, the RRT and Artificial Potential Field algorithm take care of the fast-mapping process in a higher dimensional complex environment. But, these algorithms face problems with finding the precise goal position for their probabilistic approach. Also, the potential field algorithm fails due to the cancellation of the created attractive and repulsive forces in certain critical positions in the environment, such as a curved obstacle, a long and L-shaped obstacle or finding a goal location too close to an obstacle.

VI. Solutions

Despite having these constraints these algorithms are widely used with little modifications according to the requirements.

The movement of a robot is not a problem of the algorithm. It purely depends on the mechanical design of the robot and its movement. So, it can be solved pretty easily depending on the environment the robot is going to be in and designing it likewise.

The knowledge of maps is not that important to the robot. For a turtle bot, it can use its laser scanners to scan the environment around it initially as well as at the time of movement. So, once it moves the map it knows the positions of the obstacles and these path-planning algorithms can be used easily.

Also, these laser scanners can predict the initial position of the robot on the map as it knows the positions of the nearby obstacles on the scanners.

The issue of the moving obstacles can be solved by the implementation of another obstacle avoidance function in the robot so that it needs not to implement the same algorithm every time it faces an unwanted obstacle.

We have implemented this technique as well using the laser scanner values of the Turtle Bot to avoid the obstacles in the way and simulated it through the Gazebo environment for visual representation.

All these issues are taken care of by the SLAM (simultaneous localization and mapping) algorithm which is the advanced implementation of the A* algorithm solving these limitations. Many real-world industrial autonomous robots with path planning nowadays use this path planning approach to move autonomously in any kind of known or unknown environment.

The problems with the RRT or Artificial Potential Field algorithm can be solved with more precise parameter tuning, different for different circumstances and requirements. Also, implementing other mentioned conventional approaches with these algorithms will help to improve their accuracy and preciseness and avoid their failures [15].

VII. Conclusion

Path planning is the most fundamental requirement of any autonomous robot or vehicle. As maximum of the robots in today's world is self-operative, their fundamental requirement is path-planning algorithms. Many robots are used for different kinds of work and moving in different kinds of environments. Therefore, they are required to have path-planning algorithms according to their executable works. Also, the implementation of obstacle avoidance with these path-planning algorithms helps them to perform more efficiently in any kind of map.

These algorithms mentioned above are the basis of path-planning algorithms. All the robots performing their tasks in a certain specified environment uses one of these algorithms as a base for their path-planning process. The robots or autonomous vehicles moving in a dynamic environment uses the SLAM algorithm to get rid of the existing constraints of the generic path planning algorithms and move autonomously in unknown environments.

Acknowledgement

This paper is completed with the help of different previous research works and online articles and blogs. Also, I am grateful to Dr. Sujatha R from Vellore Institute of Technology to guide me throughout my research work for this paper.

References

[1] Pablo Muñoz, María D. R-Moreno and David F. Barrero. "Unified Framework for path-planning and task-planning for autonomous robots" in Robotics and Autonomous Systems Elsevier, 30 January 2015, Revised 4 April 2016, Accepted 22 April 2016, Available online 11 May 2016, Version of Record 6 June 2016.

[2] Clement Petres; Yan Pailhas; Pedro Patron; Yvan Petillot; Jonathan Evans; David Lane, "Path Planning for Autonomous Underwater Vehicles" in IEEE Transactions on Robotics, April 2007.

[3] N. Sariff and N. Buniyamin, "An Overview of Autonomous Mobile Robot Path Planning Algorithms," in 4th Student Conference on Research and Development (Scored 2006), June 2006

[4] Christos Katrakazas, Mohammed Quddus, Weh-Hua Chen and Lipika Deka, " Real-time motion planning methods for autonomous on-road driving: State-of-the-

art and future research directions" in Transportation Research Part C 60 (2015) 416–442

[5] Buniyamin N., Wan Ngah W.A.J., Sariff N. and Mohamad Z.," A Simple Local Path Planning Algorithm for Autonomous Mobile Robots" in International Journal of Systems Applications, Engineering & Development, Issue 2, Volume 5, 2011

[6] Adem Tuncer and Mehmet Yildirim, "Dynamic path planning of mobile robots with improved genetic algorithm", in Computers and Electrical Engineering 38 (2012) 1564–1572

[7] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart and Cesar Cadena, "From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots", in 2017 IEEE International Conference on Robotics and Automation (ICRA) Singapore, May 29 - June 3, 2017

[8] Chaymaa Lamini, Said Benhlima and Ali Elbekri, " Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning " in The First International Conference On Intelligent Computing in Data Sciences.

[9] Ricardo Tellez, "The ROS Development by Construct – Path Planning" in rds the construction website blog.

[10] Eric A. Hansen and Shlomo Zilberstein, "LAO∗: A heuristic search algorithm that finds solutions with loops", in Elsevier Artificial Intelligence Volume 129, Issues 1–2, June 2001.

[11] František Duchoň, Andrej Babinec, Martin Kajan, "Path Planning with Modified a Star Algorithm for a Mobile Robot" in Procedia Engineering Volume 96, 2014

[12] Nik A. Melchior and Reid Simmons, "Particle RRT for Path Planning with Uncertainty" in 2007 IEEE International Conference on Robotics and Automation Roma, Italy, 10-14 April 2007

[13] Haojian Zhang and Yunkuan Wang, "Path Planning of Industrial Robot Based on Improved RRT Algorithm in Complex Environments" in IEEE Access

[14] Prahlad Vadakkepat, Kay Chen Tan and Wang Ming-Liang, "Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path-Planning" in IEEE Conference on Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)

[15] Hongqiang Sang, Yusong You, Xiujun Sun, Ying Zhou and Fen Liu, "The hybrid path planning algorithm based on improved A* and artificial potential field for unmanned surface vehicle formations" in Elsevier Ocean Engineering Volume 223, 1 March 2021

[16] Seyyed Mohammad Hosseini Rostami, Arun Kumar Sangaiah, Jin Wang and Xiaozhu Liu, "Obstacle avoidance of mobile robots using modified artificial potential field algorithm" in EURASIP Journal on Wireless Communications and Networking volume 2019, Article number: 70 (2019), Springer.

[17] B.K. Patel, Ganesh Babu L, Anish Pandey, D.R.K. Parhi and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot" in Elsevier Defence Technology.

[18] Siyu Guo, Xiuguo Zhang, Yisong Zheng and Yiquan Du, "An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning" in Sensors 2020, 11 January 2020.

[19] Zeyan Wu, Shaopeng Dong and Chengbin Tong, "Rotate artificial potential field algorithm toward 3D real-time path planning for unmanned aerial vehicle" in Sage Journals, August 23, 2022.

[20] Pengwei Wang, Song Gao ,Liang Li ,Binbin Sun and Shuo Cheng, "Obstacle Avoidance Path Planning Design for Autonomous Driving Vehicles Based on an Improved Artificial Potential Field Algorithm" in MDPI Energies 2019, 19 June 2019.

[21] Milad Nazarahari, Esmaeel Khanmirza, Samira Doostie, "Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm" in ScienceDirect Esevier, Expert Systems with Applications, Volume 115, January 2019